# throw manual page - Tcl Built-In Commands

tcl.tk/man/tcl/TclCmd/throw.htm

## NAME

throw — Generate a machine-readable error

## SYNOPSIS

**throw** *type message*

## DESCRIPTION

This command causes the current evaluation to be unwound with an error. The error created is described by the *type* and *message* arguments: *type* must contain a list of words describing the error in a form that is machine-readable (and which will form the error-code part of the result dictionary), and *message* should contain text that is intended for display to a human being.
The stack will be unwound until the error is trapped by a suitable **catch** or **try** command. If it reaches the event loop without being trapped, it will be reported through the **bgerror** mechanism. If it reaches the top level of script evaluation in **tclsh**, it will be printed on the console before, in the non-interactive case, causing an exit (the behavior in other programs will depend on the details of how Tcl is embedded and used).

By convention, the words in the *type* argument should go from most general to most specific.

## EXAMPLES

The following produces an error that is identical to that produced by **expr** when trying to divide a value by zero.

```
throw {ARITH DIVZERO {divide by zero}} {divide by zero}
```

# error manual page - Built-In Commands

tcl.tk/man/tcl/TclCmd/error.htm

## NAME

error — Generate an error

## SYNOPSIS

**error** *message* ?*info*? ?*code*?

## DESCRIPTION

Returns a **TCL_ERROR** code, which causes command interpretation to be unwound. *Message* is a string that is returned to the application to indicate what went wrong.
The **-errorinfo** return option of an interpreter is used to accumulate a stack trace of what was in progress when an error occurred; as nested commands unwind, the Tcl interpreter adds information to the **-errorinfo** return option. If the *info* argument is present, it is used to initialize the **-errorinfo** return options and the first increment of unwind information will not be added by the Tcl interpreter. In other words, the command containing the **error** command will not appear in the stack trace; in its place will be *info*. Historically, this feature had been most useful in conjunction with the **catch** command: if a caught error cannot be handled successfully, *info* can be used to return a stack trace reflecting the original point of occurrence of the error:

```
catch {...} errMsg
set savedInfo $::errorInfo
        ...
error $errMsg $savedInfo
```

When working with Tcl 8.5 or later, the following code should be used instead:

```
catch {...} errMsg options
        ...
return -options $options $errMsg
```

If the *code* argument is present, then its value is stored in the **-errorcode** return option. The **-errorcode** return option is intended to hold a machine-readable description of the error in cases where such information is available; see the **return** manual page for information on the proper format for this option's value.

## EXAMPLE

Generate an error if a basic mathematical operation fails:

```
if {1+2 != 3} {
    error "something is very wrong with addition"
}
```

# break manual page - Built-In Commands

tcl.tk/man/tcl/TclCmd/break.htm

## NAME

break — Abort looping command

## SYNOPSIS

**break**

## DESCRIPTION

This command is typically invoked inside the body of a looping command such as **for** or **foreach** or **while**. It returns a 3 (**TCL_BREAK**) result code, which causes a break exception to occur. The exception causes the current script to be aborted out to the innermost containing loop command, which then aborts its execution and returns normally. Break exceptions are also handled in a few other situations, such as the **catch** command, Tk event bindings, and the outermost scripts of procedure bodies.

## EXAMPLE

Print a line for each of the integers from 0 to 5:

```
for {set x 0} {$x<10} {incr x} {
    if {$x > 5} {
        break
    }
    puts "x is $x"
}
```

# continue manual page - Built-In Commands

tcl.tk/man/tcl/TclCmd/continue.htm

## NAME

continue — Skip to the next iteration of a loop

## SYNOPSIS

**continue**

## DESCRIPTION

This command is typically invoked inside the body of a looping command such as **for** or **foreach** or **while**. It returns a 4 (**TCL_CONTINUE**) result code, which causes a continue exception to occur. The exception causes the current script to be aborted out to the innermost containing loop command, which then continues with the next iteration of the loop. Catch exceptions are also handled in a few other situations, such as the **catch** command and the outermost scripts of procedure bodies.

## EXAMPLE

Print a line for each of the integers from 0 to 10 *except* 5:

```
for {set x 0} {$x<10} {incr x} {
    if {$x == 5} {
        continue
    }
    puts "x is $x"
}
```